

## Lab 9

**Reg. No: 19BCE1209**

**Name: Gautam Sanjay Wadhwani**

**Course: CSE4001 Parallel and Distributed Computing**

### **Q1. Circuit Satisfiability**

**Code:**

```
#include <mpi.h>
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
int check(int* a, int n)
```

```
{
```

```
    if((a[0]&a[1]) | ((!a[0])&!a[1])) | (a[0]&a[2]))
```

```
    {
```

```
        return 1;
```

```
    }
```

```
    else
```

```
    {
```

```
        return 0;
```

```
    }
```

```
}
```

```
void generate_bits(int* a, int x, int n)
```

```
{
```

```
    while(n-- > 0)
```

```
    {
```

```
        a[n] = x&1;
```

```
        x >>= 1;
```

```
    }
```

```
}
```

```

int main(int argc, char** argv)
{
    int p, id, num_vars = 3, satisfiable = 0;
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    MPI_Comm_rank(MPI_COMM_WORLD, &id);
    MPI_Request request[p];
    MPI_Status status;
    int n = 1 << num_vars;
    for(int x = id; x < n; x += p)
    {
        int a[num_vars];
        generate_bits(a, x, num_vars);
        if(check(a, num_vars))
        {
            satisfiable = 1;
            break;
        }
    }
    if(id == 0)
    {
        for(int i = 1; i < p; i++)
        {
            int x;
            MPI_Irecv(&x, 1, MPI_INT, i, 123, MPI_COMM_WORLD, &request[i]);
            MPI_Wait(&request[i], &status);
            if(x) printf("Received satisfiable = %d from process %d\n", x, i);
            satisfiable |= x;
        }
        if(satisfiable) printf("The circuit is satisfiable\n");
        else printf("The circuit is not satisfiable\n");
    }
}

```

```

    }
else
{
    printf("Process %d: sending satisfiable = %d\n", id, satisfiable);
    MPI_Isend(&satisfiable, 1, MPI_INT, 0, 123, MPI_COMM_WORLD, &request[id]);
    MPI_Wait(&request[id], &status);
}
MPI_Finalize();
return 0;
}

```

### Output:

Let circuit be:  $(x \wedge y) \vee (!x \wedge !y) \vee (x \wedge z)$

```

gautam@ubuntu:~$ mpicc lab_9_1.c -o pdc
gautam@ubuntu:~$ mpirun -np 4 ./pdc
Process 2: sending satisfiable = 1
Process 3: sending satisfiable = 1
Process 1: sending satisfiable = 1
Received satisfiable = 1 from process 1
Received satisfiable = 1 from process 2
Received satisfiable = 1 from process 3
The circuit is satisfiable
gautam@ubuntu:~$

```

Let circuit be:  $(!x \vee !y) \wedge (x \wedge y)$

```

gautam@ubuntu:~$ mpicc lab_9_1.c -o pdc
gautam@ubuntu:~$ mpirun -np 4 ./pdc
Process 1: sending satisfiable = 0
Process 2: sending satisfiable = 0
Process 3: sending satisfiable = 0
The circuit is not satisfiable
gautam@ubuntu:~$

```

### Q2. Number of solutions in circuit satisfiability

#### Code:

```

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

```

```

int check(int* a, int n)
{
    if((a[0]&a[1]) | ((!a[0])&!a[1])) | (a[0]&a[2]))
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

void generate_bits(int* a, int x, int n)
{
    while(n-- > 0)
    {
        a[n] = x&1;
        x >>= 1;
    }
}

int main(int argc, char** argv)
{
    int p, id, num_vars = 3, satisfiable = 0;
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    MPI_Comm_rank(MPI_COMM_WORLD, &id);
    MPI_Request request[p];
    MPI_Status status;
    int n = 1 << num_vars;
    for(int x = id; x < n; x += p)
    {
        int a[num_vars];

```

```

    generate_bits(a, x, num_vars);
    satisfiable += check(a, num_vars);
}
if(id == 0)
{
    for(int i = 1; i < p; i++)
    {
        int x;
        MPI_Irecv(&x, 1, MPI_INT, i, 120+i, MPI_COMM_WORLD, &request[i]);
        MPI_Wait(&request[i], &status);
        printf("Received satisfiable = %d from process %d\n", x, i);
        satisfiable += x;
    }
    printf("The circuit is has %d solutions\n", satisfiable);
}
else
{
    printf("Process %d: sending satisfiable = %d\n", id, satisfiable);
    MPI_Isend(&satisfiable, 1, MPI_INT, 0, 120+id, MPI_COMM_WORLD, &request[id]);
    MPI_Wait(&request[id], &status);
}
MPI_Finalize();
return 0;
}

```

**Output:**

Let circuit be:  $(x \wedge y) \vee (!x \wedge !y) \vee (x \wedge z)$

```

gautam@ubuntu:~$ mpicc lab_9_2.c -o pdc
gautam@ubuntu:~$ mpirun -np 4 ./pdc
Process 3: sending satisfiable = 1
Process 1: sending satisfiable = 2
Received satisfiable = 2 from process 1
Process 2: sending satisfiable = 1
Received satisfiable = 1 from process 2
Received satisfiable = 1 from process 3
The circuit is has 5 solutions
gautam@ubuntu:~$

```

Let circuit be:  $(!x \vee !y) \wedge (x \wedge y)$

```

gautam@ubuntu:~$ mpicc lab_9_2.c -o pdc
gautam@ubuntu:~$ mpirun -np 4 ./pdc
Process 1: sending satisfiable = 0
Process 3: sending satisfiable = 0
Received satisfiable = 0 from process 1
Received satisfiable = 0 from process 2
Process 2: sending satisfiable = 0
Received satisfiable = 0 from process 3
The circuit is has 0 solutions
gautam@ubuntu:~$

```

**Q3.** Adding a count to all values of a matrix with size  $n \times n$

**Code:**

```
#include <mpi.h>
```

```
#include <stdio.h>
```

```

int main(int argc, char** argv) {
    int p, id, n = 3, count = 1;

    int mat[3][3] = {{0, 1, 2}, {1, 2, 3}, {4, 3, 1}};

    MPI_Init(NULL, NULL);

    MPI_Comm_size(MPI_COMM_WORLD, &p);
    MPI_Comm_rank(MPI_COMM_WORLD, &id);

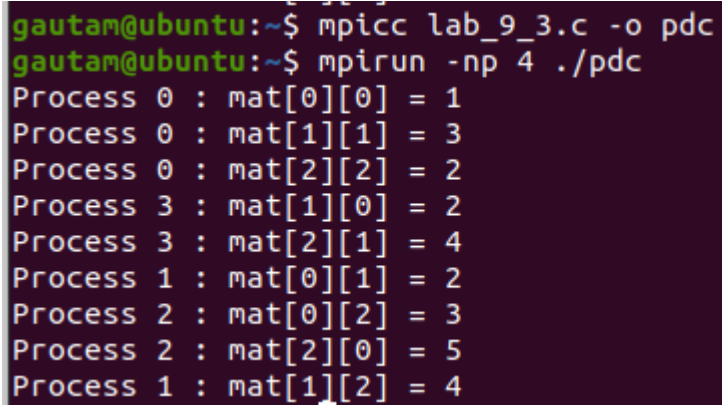
    for(int x = id; x < n*n; x += p)
    {
        int i = x / n, j = x % n;
        mat[i][j] += count;

        printf("Process %d : mat[%d][%d] = %d\n", id, i, j, mat[i][j]);
    }
}

```

```
MPI_Finalize();  
  
return 0;  
  
}
```

**Output:**



A terminal window showing the compilation and execution of an MPI program. The user 'gautam' is at a 'ubuntu' machine. They compile 'lab\_9\_3.c' with 'mpicc' to create 'pdc'. Then they run 'pdc' with 'mpirun -np 4'. The output shows four processes (0, 1, 2, 3) each printing a value from a matrix 'mat'.

```
gautam@ubuntu:~$ mpicc lab_9_3.c -o pdc  
gautam@ubuntu:~$ mpirun -np 4 ./pdc  
Process 0 : mat[0][0] = 1  
Process 0 : mat[1][1] = 3  
Process 0 : mat[2][2] = 2  
Process 3 : mat[1][0] = 2  
Process 3 : mat[2][1] = 4  
Process 1 : mat[0][1] = 2  
Process 2 : mat[0][2] = 3  
Process 2 : mat[2][0] = 5  
Process 1 : mat[1][2] = 4
```

**Q4.** Find Max of 'n' no's

**Code:**

```
#include <mpi.h>  
  
#include <stdio.h>  
  
#include <math.h>  
  
#include <limits.h>  
  
  
int main(int argc, char** argv) {  
  
    int p, id, n = 10;  
  
    int arr[10] = {1, 3, 5, 4, 6, 8, 2, 1, 0, 3};  
  
    MPI_Request request;  
  
    MPI_Status status;  
  
    MPI_Init(NULL, NULL);  
  
    MPI_Comm_size(MPI_COMM_WORLD, &p);  
  
    MPI_Comm_rank(MPI_COMM_WORLD, &id);  
  
    int m = INT_MIN;  
  
    for(int i = id; i < n; i += p)  
    {  
  
        if(arr[i] > m) m = arr[i];  
  
    }  
  
}
```

```

if(id != 0)
{
    printf("Process: %d sending max = %d\n", id, m);
    MPI_Isend(&m, 1, MPI_INT, 0, 123, MPI_COMM_WORLD, &request);
    MPI_Wait(&request, &status);
}
else
{
    printf("Process: 0 max = %d\n", m);
    for(int j = 1; j < p; j++)
    {
        int x;
        MPI_Request requestj;
        MPI_Irecv(&x, 1, MPI_INT, j, 123, MPI_COMM_WORLD, &requestj);
        MPI_Wait(&requestj, &status);
        printf("Received max = %d from process: %d\n", x, j);
        if(x > m) m = x;
    }
    printf("Max of array is : %d\n", m);
}
MPI_Finalize();
return 0;
}

```

#### Output:

```

gautam@ubuntu:~$ mpicc lab_9_4.c -o pdc
gautam@ubuntu:~$ mpirun -np 4 ./pdc
Process: 0 max = 6
Process: 1 sending max = 8
Process: 3 sending max = 4
Received max = 8 from process: 1
Received max = 5 from process: 2
Received max = 4 from process: 3
Max of array is : 8
Process: 2 sending max = 5

```



#### Q5. Four Queen's Problem

##### Code:

```
#include <mpi.h>

#include <stdio.h>

int factorial(int n)
{
    int f = 1;
    for(int i = 2; i < n+1; i++) f *= i;
    return f;
}

int check(int n, int k, int id)
{
    // let kth permutation of [0...n-1] = perm[n]

    // let coordinates of ith queen = (i, perm[i]), so that only one queen in each row and only one
    queen in each column

    int len = n;
    int perm[n];
    int nums[n];
    for(int i = 0; i < n; i++) nums[i] = i;
    int size = n;
    for(int i = 0; i < n; i++)
    {
        int index = k / factorial(n-i-1);
        perm[i] = nums[index]; // ith element of the kth permutation of [0...n-1]
        for(int j = index; j < size-1; j++) nums[j] = nums[j+1]; size--; // delete nums[index]
        k -= index*factorial(n-i-1);
    }

    // check if two queens on same diagonal or anti-diagonal

    // same diagonal -> same i-j value
    // same anti-diagonal -> same i+j value
    // coordinates of ith queen = (i, perm[i])
```

```

for(int i = 0; i < n; i++)
{
    for(int j = i+1; j < n; j++)
    {
        // check if ith queen and jth queen on same diagonal or anti-diagonal
        if(i-perm[i] == j-perm[j] || i+perm[i] == j+perm[j])
        {
            return 0; // invalid arrangement
        }
    }
}

printf("Process %d: found solution place queens at: (%d, %d) (%d, %d) (%d, %d) (%d, %d)\n", id, 0,
perm[0], 1, perm[1], 2, perm[2], 3, perm[3]);

return 1; // valid arrangement
}

int main(int argc, char** argv) {
    int p, id, n = 4;
    int f = factorial(n);
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    MPI_Comm_rank(MPI_COMM_WORLD, &id);
    for(int x = id; x < f; x += p)
    {
        check(n, x, id);
    }
    MPI_Finalize();
    return 0;
}

```

**Output:**

```

gautam@ubuntu:~$ mpicc lab_9_5.c -o pdc
gautam@ubuntu:~$ mpirun -np 4 ./pdc
Process 1: found solution place queens at: (0, 2) (1, 0) (2, 3) (3, 1)
Process 2: found solution place queens at: (0, 1) (1, 3) (2, 0) (3, 2)
gautam@ubuntu:~$

```

**Q6.** Sample isend, ireceive with mpi\_wtime

**Code:**

```

#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv)
{
    int rank;
    int buffer;
    int world_size;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    MPI_Status status;
    MPI_Request request;
    double time = 0;

    if(rank == 0)
    {
        buffer = 10;
        printf("Rank %d sending %d\n", rank, buffer);
        time = - MPI_Wtime();
        for(int i = 0; i < world_size; i++)
        {
            MPI_Isend(&buffer, 1, MPI_INT, i, 123, MPI_COMM_WORLD, &request);
        }
        time += MPI_Wtime();
    }
}

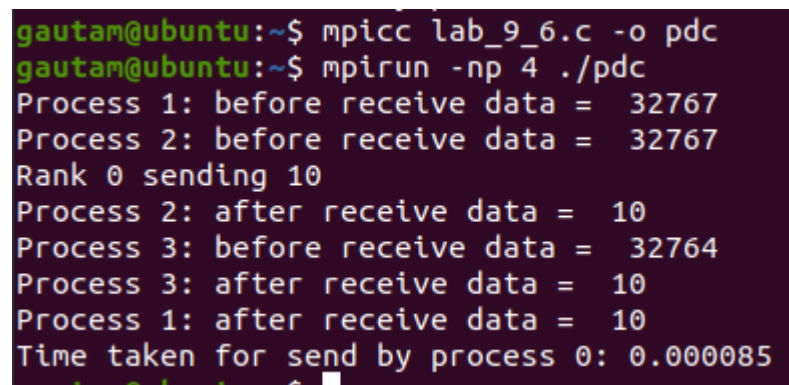
```

```

else
{
    printf("Process %d: before receive data = %d\n", rank, buffer);
    MPI_Irecv(&buffer, 1, MPI_INT, 0, 123, MPI_COMM_WORLD, &request);
    MPI_Wait(&request, &status);
    printf("Process %d: after receive data = %d\n", rank, buffer);
}
MPI_Finalize();
if(rank == 0)
{
    printf("Time taken for send by process %d: %f\n", rank, time);
}
return 0;
}

```

#### Output:



```

gautam@ubuntu:~$ mpicc lab_9_6.c -o pdc
gautam@ubuntu:~$ mpirun -np 4 ./pdc
Process 1: before receive data = 32767
Process 2: before receive data = 32767
Rank 0 sending 10
Process 2: after receive data = 10
Process 3: before receive data = 32764
Process 3: after receive data = 10
Process 1: after receive data = 10
Time taken for send by process 0: 0.000085
gautam@ubuntu:~$

```

#### Q7. Sample send and receive with mpi\_wtime

##### Code:

```

#include <mpi.h>
#include <stdio.h>
int main(int argc, char** argv)
{
    int rank;
    int buffer;
    int world_size;

```

```

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
double time = 0;

if(rank == 0)
{
    buffer = 10;
    printf("Rank %d sending %d\n", rank, buffer);
    time = - MPI_Wtime();
    for(int i = 0; i < world_size; i++)
    {
        MPI_Send(&buffer, 1, MPI_INT, i, 123, MPI_COMM_WORLD);
    }
    time += MPI_Wtime();
}
else
{
    printf("Process %d: before receive data = %d\n", rank, buffer);
    MPI_Recv(&buffer, 1, MPI_INT, 0, 123, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("Process %d: after receive data = %d\n", rank, buffer);
}
MPI_Finalize();
if(rank == 0)
{
    printf("Time taken for send by process %d: %f\n", rank, time);
}
return 0;
}

```

**Output:**

```

gautam@ubuntu:~$ mpicc lab_9_7.c -o pdc
gautam@ubuntu:~$ mpirun -np 4 ./pdc
Rank 0 sending 10
Process 2: before receive data = 0
Process 2: after receive data = 10
Process 3: before receive data = 0
Process 3: after receive data = 10
Process 1: before receive data = 0
Process 1: after receive data = 10
Time taken for send by process 0: 0.000197

```

**Q8.** Implementing the broadcast using send and receive

**Code:**

```
#include<stdio.h>
```

```
#include<mpi.h>
```

```
void my_bcast(void* data, int count, MPI_Datatype datatype, int root, MPI_Comm communicator)
```

```
{
```

```
    int rank;
```

```
    MPI_Comm_rank(communicator, &rank);
```

```
    int world_size;
```

```
    MPI_Comm_size(communicator, &world_size);
```

```
    if(rank == root)
```

```
    {
```

```
        int i;
```

```
        for(i = 0; i < world_size; i++)
```

```
        {
```

```
            if(i != rank)
```

```
            {
```

```
                MPI_Send(data, count, datatype, i, 0, communicator);
```

```
            }
```

```
        }
```

```
    }
```

```
    else
```

```
    {
```

```
        MPI_Recv(&data, count, datatype, root, 0, communicator, MPI_STATUS_IGNORE);
```

```

        printf("Rank %d received data = %d\n", rank, data);
    }
}

int main()
{
    MPI_Init(NULL, NULL);

    int data = 10;

    int data_count = 1;

    int rank;

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

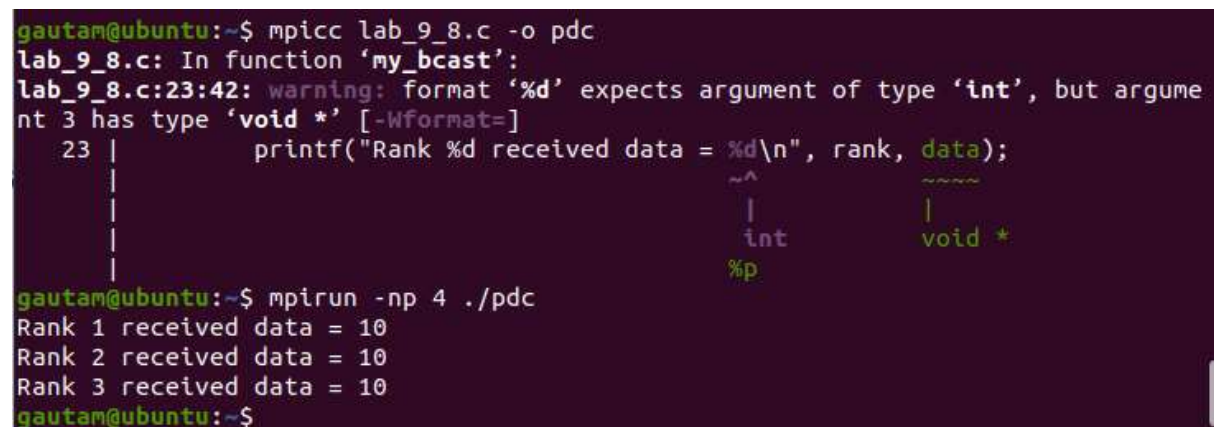
    my_bcast(&data, data_count, MPI_INT, 0, MPI_COMM_WORLD);

    MPI_Finalize();

    return 0;
}

```

#### Output:



```

gautam@ubuntu:~$ mpicc lab_9_8.c -o pdc
lab_9_8.c: In function 'my_bcast':
lab_9_8.c:23:42: warning: format '%d' expects argument of type 'int', but argume
nt 3 has type 'void *' [-Wformat=]
   23 |         printf("Rank %d received data = %d\n", rank, data);
      |                                ~^      ~~~~
      |                                |      |
      |                                int   void *
      |                                %p
gautam@ubuntu:~$ mpirun -np 4 ./pdc
Rank 1 received data = 10
Rank 2 received data = 10
Rank 3 received data = 10
gautam@ubuntu:~$

```

#### Q9. Ring communication

##### Code:

```

#include <mpi.h>

#include <stdio.h>

int main(int argc, char *argv[])
{
    int id, p, left, right, n = 10;

    MPI_Request request, request2;

```

```

MPI_Status status;

MPI_Init(&argc,&argv);

MPI_Comm_size(MPI_COMM_WORLD, &p);

MPI_Comm_rank(MPI_COMM_WORLD, &id);


right = (id + 1) % p;

left = id - 1;

if(left < 0) left = p - 1;


MPI_Isend(&n, 1, MPI_INT, right, 123, MPI_COMM_WORLD, &request2);

MPI_Wait(&request2, &status);

MPI_Irecv(&n, 1, MPI_INT, left, 123, MPI_COMM_WORLD, &request);

MPI_Wait(&request, &status);


printf("Process %d: received n = %d from process %d\n", id, n, left);

printf("Process %d: sending n = %d to process %d\n", id, n, right);

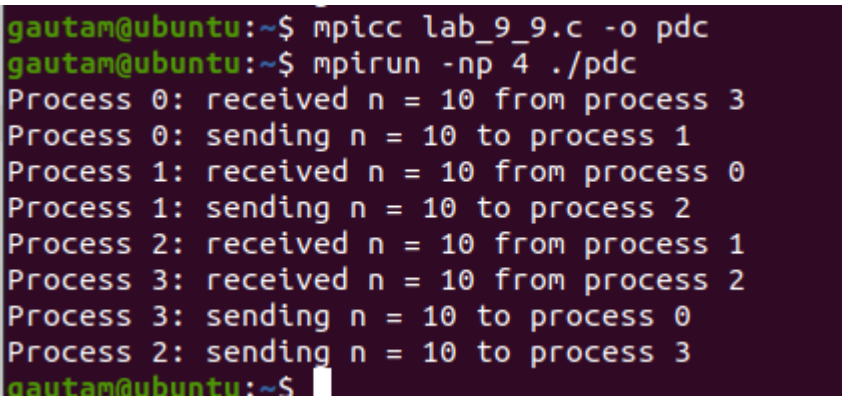
MPI_Finalize();

return 0;

}

```

#### Output:



```

gautam@ubuntu:~$ mpicc lab_9_9.c -o pdc
gautam@ubuntu:~$ mpirun -np 4 ./pdc
Process 0: received n = 10 from process 3
Process 0: sending n = 10 to process 1
Process 1: received n = 10 from process 0
Process 1: sending n = 10 to process 2
Process 2: received n = 10 from process 1
Process 3: received n = 10 from process 2
Process 3: sending n = 10 to process 0
Process 2: sending n = 10 to process 3
gautam@ubuntu:~$

```

**Q10.** rank0 - sends randnum, rank1 - Add const , rank2 - sub const, rank3 - mul const

#### Code:

```

#include <mpi.h>

#include <stdio.h>

```



```

int main(int argc, char** argv) {

    int p, id, randnum, constant = 2;

    MPI_Status status;

    MPI_Init(NULL, NULL);

    MPI_Comm_size(MPI_COMM_WORLD, &p);

    MPI_Comm_rank(MPI_COMM_WORLD, &id);

    if(id == 0)
    {
        randnum = 10;

        printf("Broadcasting randnum = %d\n", randnum);

        MPI_Bcast(&randnum, 1, MPI_INT, 0, MPI_COMM_WORLD);

        int x;

        MPI_Request request1, request2, request3;

        MPI_Irecv(&x, 1, MPI_INT, 1, 111, MPI_COMM_WORLD, &request1);

        MPI_Wait(&request1, &status);

        printf("Received %d from process: 1\n", x);

        MPI_Irecv(&x, 1, MPI_INT, 2, 112, MPI_COMM_WORLD, &request2);

        MPI_Wait(&request2, &status);

        printf("Received %d from process: 2\n", x);

        MPI_Irecv(&x, 1, MPI_INT, 3, 113, MPI_COMM_WORLD, &request3);

        MPI_Wait(&request3, &status);

        printf("Received %d from process: 3\n", x);

    }

    else if(id == 1)
    {

        MPI_Bcast(&randnum, 1, MPI_INT, 0, MPI_COMM_WORLD);

        MPI_Request request;

        int x = randnum + constant;

        MPI_Isend(&x, 1, MPI_INT, 0, 111, MPI_COMM_WORLD, &request);

        MPI_Wait(&request, &status);
    }
}

```

```

}
else if(id == 2)
{
    MPI_Bcast(&randnum, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Request request;
    int x = randnum - constant;
    MPI_Isend(&x, 1, MPI_INT, 0, 112, MPI_COMM_WORLD, &request);
    MPI_Wait(&request, &status);
}
else if(id == 3)
{
    MPI_Bcast(&randnum, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Request request;
    int x = randnum * constant;
    MPI_Isend(&x, 1, MPI_INT, 0, 113, MPI_COMM_WORLD, &request);
    MPI_Wait(&request, &status);
}
if(id != 0)
{
    printf("Process %d: Received from broadcast randnum = %d\n", id, randnum);
}
MPI_Finalize();
return 0;
}

```

#### Output:

```

gautam@ubuntu:~$ mpicc lab_9_10.c -o pdc
gautam@ubuntu:~$ mpirun -np 4 ./pdc
Broadcasting randnum = 10
Process 1: Received from broadcast randnum = 10
Process 2: Received from broadcast randnum = 10
Received 12 from process: 1
Received 8 from process: 2
Process 3: Received from broadcast randnum = 10
Received 20 from process: 3
gautam@ubuntu:~$

```

**Q11.** Simulate a chat window - server answers query to client

**Code:**

ChatI.java:

```
import java.rmi.Remote;
```

```
import java.rmi.RemoteException;
```

```
public interface ChatI extends Remote {  
    public String reply(String query) throws RemoteException;  
}
```

Chat.java:

```
import java.rmi.RemoteException;
```

```
import java.rmi.server.UnicastRemoteObject;
```

```
public class Chat extends UnicastRemoteObject implements ChatI {  
    public Chat() throws RemoteException {  
        super();  
    }  
  
    public String reply(String query) throws RemoteException{  
        return "Server: reverse = " + new String((new StringBuilder(query)).reverse());  
    }  
}
```

Client.java

```
import java.rmi.RemoteException;
```

```
import java.rmi.Naming;
```

```
import java.util.Scanner;
```

```
public class Client {  
    public Client() {  
  
    }  
}
```

```

public static void main(String[] args) throws RemoteException {

    Scanner sc = new Scanner(System.in);

    System.out.print("Client: ");

    String message = sc.next();

    Chat chat = null;

    try {

        chat = (Chat) Naming.lookup("rmi://localhost:5000/chat");

    } catch (Exception e) {

        e.printStackTrace();

    }

    System.out.println(chat.reply(message));

    sc.close();

}
}

```

Server.java

```

import java.rmi.RemoteException;

import java.rmi.Naming;

```

```

public class Server {

    public Server() throws RemoteException {

        super();

    }

    public static void main(String[] args) throws RemoteException {

        Chat chat = new Chat();

        try {

            Naming.rebind("rmi://localhost:5000/chat", chat);

        } catch (Exception e) {

            e.printStackTrace();

        }

    }
}

```

```
        System.out.println("Server is waiting...");
    }
}
```

#### Output:

```
gautam@ubuntu:~/LAB 9 RMI$ javac Server.java
gautam@ubuntu:~/LAB 9 RMI$ javac Client.java
gautam@ubuntu:~/LAB 9 RMI$ rmic Chat
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
gautam@ubuntu:~/LAB 9 RMI$ rmiregistry 5000
```

```
gautam@ubuntu:~/LAB 9 RMI$ java Server
Server is waiting...
```

```
gautam@ubuntu:~/LAB 9 RMI$ javac Client.java
gautam@ubuntu:~/LAB 9 RMI$ java Client
Client: Hello
Server: reverse = olleH
gautam@ubuntu:~/LAB 9 RMI$
```